Electronic Communications of the EASST
Volume 6 (2007)

Proceedings of the Sixth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)

Imposing Hierarchy on a Graph

Brendan Sheehan, Benoit Gaudin and Aaron Quigley

15 pages

Volume Editor: Karsten Ehrig, Holger Giese
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Imposing Hierarchy on a Graph

Brendan Sheehan, Benoit Gaudin and Aaron Quigley

Systems Research Group
University College Dublin, Belfield, Dublin 4, Ireland
{Brendan.Sheehan, Benoit.Gaudin, Aaron.Quigley}@ucd.ie

Abstract: This paper investigates a way of imposing a hierarchy on a graph in order to explore relationships between elements of data. Imposing a hierarchy is equivalent to clustering. First a tree structure is imposed on the initial graph, then a *k*-partite structure is imposed on each previously obtained cluster. Imposing a tree exposes the hierarchical structure of the graph as well as providing an abstraction of the data. In this study three kinds of merge operations are considered and their composition is shown to yield a tree with a maximal number of vertices in which vertices in the tree are associated with disjoint connected subgraphs. These subgraphs are subsequently transformed into k-partite graphs using similar merge operations. These merges also ensure that the obtained tree is proper with respect to the hierarchy imposed on the data.

A detailed example of the technique's application in exposing the structure of protein interaction networks is described. The example focuses on the MAPK cell signalling pathway. The merge operations help expose where signal regulation occurs within the pathway and from other signalling pathways within the cell.

Keywords: Graph visualization, clustering technique, tree, *k*-partite graphs.

1 Introduction

When exploring the structure of large graphs the user is often hindered by the number of vertices and edge crossings contained in an unabstracted visualization. This study deals with an approach to transforming the underlying graph with respect to a well understood structure namely a tree structure. The user can then explore the graph with respect to this structure. Vertices in the underlying graph are clustered together into super-vertices by edge contraction in such a way that a tree structure emerges. The question that must then be addressed is what edges are contracted and in what order to result in such a structure and how does the abstract tree relate to the underlying graph in such a way that it can be easily interpreted by the user.

The idea of abstracting a graph in order to improve its interpretability is not new. Each approach to abstraction is designed to achieve different goals. In [Fen97], for example, the authors cluster vertices to obtain a planar abstraction of the initial graph. The obtained graph does then not contain any cross-edges thus improving the readability. When exploring a graph without any particular question in mind planarity is a good criterion to impose. However some graphs possess an underlying structure (such as a tree or grid structure) which the user would like to expose in order to address particular questions. In [HHP05], for example, the authors show that

users prefer hierarchical graphs when understanding the data set. For this reason, abstracting the graph as a tree is considered in this study.

Extracting a tree from a graph is usually done by means of *spanning tree* computation techniques. But spanning trees only partially achieve our goals. The number of nodes of a spanning tree is the same as the number of vertices of the graph. There have been attempts to visualize large networks using spanning trees [LKK06] but ultimately such layouts do not minimize the number of vertices that the user has to read.

There exist other methods to expose the tree-like structure of a graph such as tree-decomposition [Die05, Bod93]. However the mapping of vertices to clusters is not one to one. This means that it is hard for the user to associate underlying vertices with such a decomposition. However the related area of *graph minor theory* which is concerned with studying the properties of graphs obtained by edge contraction or edge deletion is of interest. Edge contraction provides a reasonable way of clustering a graph in such a way that it remains comprehensible to the user. As shown in [DK05] edge contraction does not have to be performed explicitly but can be divided into two separate sets of operations to obtain a *pre-minor* and *minor* of the underlying graph respectively. Such operations are termed *merges* in this paper.

Intuitively, a merge collapses some vertices of the initial graph into clusters and updates the edges so that two clusters are connected if they share a common edge. Three merges are considered in this paper. The first one, introduced in [BM04], yields a tree whose number of vertices is maximal. This tree represents an abstraction of the initial graph, enabling the user to browse the smaller graph. This is of particular relevance when the initial graph is large. If one of the vertices of the tree appears to be of interest then the user may want to see the part of the initial graph to which it corresponds. Often it is desirable that clusters of the graph are connected. Since this is not ensured by the merge previously mentioned, a second merge operation is considered, which has to be applied together with the first merge operation (the two merge operations have to be composed). Finally, the third merge introduced enables visualization of the contents of clusters as k-partite graphs.

In section 2, some notation and preliminaries are introduced. The first merge operation is presented in section 3, the second merge operation in section 4, and the third merge is introduced in section 5. It is shown in these sections that their composition yields a tree with maximal number of vertices. The nodes of this tree are obtained by clustering and the contents of a cluster is itself cluster to result in a k-partite graph. Finally, an example of the algorithms application is provided based on protein-protein interaction data provided by CPATH [cpa] in section 6.

2 Preliminaries

A simple graph G is a couple (V_G, E_G) where V_G is the set of vertices of G and E_G is its set of edges, such that $\{(x,x') \in V_G^2 | x = x'\} \subseteq E_G \subseteq V_G^2$. A graph G is said to be empty if $V_G = \emptyset$. Finally, G is said to be undirected if for all $(x,y) \in E_G$, it is also true that $(y,x) \in E_G$. The set of undirected simple graphs is denoted $\mathbf{G}$. In what follows, only undirected graphs will be considered.

Given a non-empty graph G and two vertices x,y of G, if $(x,y) \in E_G$ then x and y are said to be neighbors. Moreover, a path p of G between x and y is a sequence of edges $(x_i, x_{i+1})_{1 \leq i \leq k}$ such

that $x = x_1$, $y = x_{k+1}$ and for all $i \in \{1,\ldots,k\}$, $(x_i, x_{i+1}) \in E_G$. The empty path is denoted ε and represents the empty sequence of edges.

If $p = (x_i, x_{i+1})_{1 \leq i \leq k}$ is a path of G, then any sub-sequence $(x_i, x_{i+1})_{k_1 \leq i \leq k_2}$ with $k_1 \geq 1$ and $k_2 \leq k$ is called a sub-path of p and is denoted $p_{x_{k_1}, x_{k_2}}$. For instance, if p equals

$$(x_1,x_2),(x_2,x_3),(x_3,x_4),(x_5,x_6),(x_6,x_7),(x_8,x_9)$$

then p_{x_3,x_6} equals $(x_3,x_4),(x_5,x_6)$. If p and p' are two paths and p (resp. p') is empty, then the concatenation of p and p', denoted $p.p'$, equals p' (resp. p). Now if neither p nor p' is empty and $p = ((x_i, x_{i+1}))_{1 \leq i \leq k}$ and $p' = ((x'_j, x'_{j+1}))_{1 \leq j \leq k'}$ and $x_{k+1} = x'_1$, then $p.p'$ is the path

$$(x_1,x_2).(x_2,x_3)\ldots(x_k,x_{k+1}).(x'_1,x'_2)\ldots(x'_{k'},x'_{k'+1})$$

The length of a path p, denoted $|p|$, is defined as the following:

$$|p| = \begin{cases} 0 & \text{if } p = \varepsilon \\ |p'| & \text{if } p = (x,x').p' \text{ with } x = x' \\ |p'|+1 & \text{if } p = (x,x').p' \text{ with } x \neq x' \end{cases}$$

For instance, considering the path p defined above, $|p| = 6$ and $|(x_1,x_1).p| = 6$ also.

Given a connected simple undirected graph G and a vertex r of G, the distance function $d_{G,r}(\cdot)$ is defined for all vertices x of G by

$$d_{G,r}(x) = \min\{|p| \in \mathbb{N} |\ p \text{ is a path in } G \text{ between } r \text{ and } x\} \tag{1}$$

As an example, for all vertices n of graph G given in Figure 1, the value of $d_{G,r}(n)$ is written down on the left side of Figure 1.

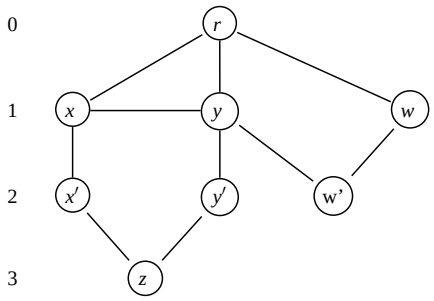
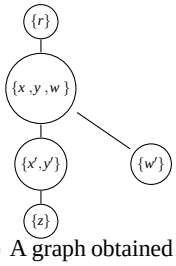


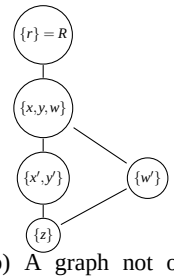
Figure 1: A graph G.

A graph is said to be k-partite if it is possible to partition vertices into k subsets such that two vertices belonging to the same partition are not neighbors. The goal of this study is to compute a tree and k-partite graphs with maximal number of vertices, by transforming a graph G. The kind of transformations under consideration are called *merges*:

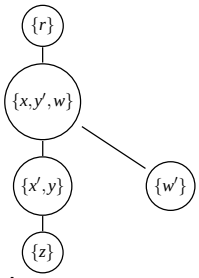
Definition 1 A merge is an operator from **G** to **G** which possesses the following properties: if $G' \in \mathbf{G}$ is the result of a merge applied to $G \in \mathbf{G}$, then



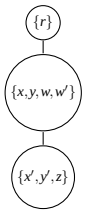
(a) A graph obtained by merging.



(b) A graph not obtained by merging.



(c) A merge not preserving the hierarchy.



(d) The tree resulting from applying M_1 and M_2.

Figure 2:

(a) $V_{G'}$ is a partition of V_G (i.e $V_{G'} \subseteq 2^{V_G}$ and $\bigcup_{X \in V_{G'}} X = V_G$ and $\forall X, Y \in V_{G'}, X \cap Y = \emptyset$).

(b) for $X, Y \in V_{G'}$,

$$(X,Y) \in E_{G'} \Longleftrightarrow \exists x \in X, y \in Y, \text{s.t}\,(x,y) \in E_G$$

Thus a merge is an operator which collapses together several vertices of a graph to obtain a new vertex. The edges incident to the collapsed vertices then incident to the newly obtained vertex. Note that a graph obtained by merging, using the first merge operation, is different from a graph minor. A merge corresponds to the repeated applications of vertex-contractions, and a minor corresponds the repeated applications of edge-contractions and edge-deletions.

Figure 3(a) represents a graph obtained by merging from the graph G of Figure 1. But Figure 3(b) represents a graph which cannot be obtained from G by merging. There is no edge between w' and z in G, and there is one between $\{w'\}$ and $\{z\}$.

Considering a particular vertex r of a graph G as well as the distance function $d_{G,r}$, a hierarchy is explicitly introduced over the vertices of G in order to facilitate the understanding of the data by the user: r is the higher element in the hierarchy and the further from r a vertex is the lower it is in the hierarchy[1]. It is usually desirable that a merge applied to G *preserves the hierarchy*. Formally, M *preserves the hierarchy* of G according to r if $\forall x, y \in V_G, \forall X, Y \in V_{M(G)}$ s.t $x \in X$ and $y \in Y$,

$$d_{G,r}(x) \leq d_{G,r}(y) \Rightarrow d_{M(G),R}(X) \leq d_{M(G),R}(Y) \tag{2}$$

where R is the vertex of $M(G)$ containing r. The graph G' represented in Figure 3(a) is obtained by a merge preserving the hierarchy of G (see Figure 1) according to r. This is not the case for the merge resulting in the graph of Figure 3(c). If R denotes the vertex $\{r\}$, then $d_{G',R}(\{x,y',w\}) < d_{G',r}(\{x',y\})$ and $d_{G,r}(x') < d_{G,r}(y')$. This means that x' is lower than y' in the hierarchy involved by G', although it is the opposite in G. In the case of a social network for instance, a merge like the one yielded in G' would then reverse the fact that the person x' plays a more important role than the person y'. This seems awkward in some situations. For this reason, merges avoiding this kind of reversion are considered in this study.

[1] Note that if $d_{G,r}(x)$ is greater than $d_{G,r}(y)$ and $d_{G,r}(y)$ is greater than $d_{G,r}(x)$ then $x = y$ does not hold in general.

Merging vertices of a graph represents an interesting way to provide an abstraction. If this abstraction is done in a reasonable way, merging could provide a basis by which to browse the graph. Indeed one could browse the result of a merge and decide to enter into details by browsing the part of the initial graph represented by a vertex obtained by merging. Such a part of the initial graph is called an *associated graph*.

Definition 2 Let G be a graph and M be a merge. Given a vertex X of $M(G)$, the graph associated to X according to G, denoted $G'(X)$ represents the sub-graph G' of G defined by:

- $V_{G'} = \{x \in V_G | x \in X\}$
- $E_{G'} = \{(x,y) \in E_G | x \in X \wedge y \in X\}$

Even if G is a connected graph, this property does not hold for any associated graph. But when considering entering into details of a vertex of a graph obtained by merging, it is often preferable that a connected sub-graph of G is associated with it. For this reason, we will consider in section 4 merges which ensure that all associated sub-graphs are connected. In what follows, such a graph will be called a *deeply connected graph* according to G.

Lemma 1 is an interesting lemma showing that the existence of path in a graph G is preserved by merging.

Lemma 1 *Let G and G' be two graphs such that G' is obtained from G by applying a merge operator. Let $x,y \in V_G$ and $X,Y \in V_{G'}$ such that $x \in X$ and $y \in Y$. If there exists a path in G between x and y, then there exists a path in G' between X and Y.*

Section 3 aims at providing a means of imposing an optimal tree from any undirected connected simple and non empty graph. Section 4 aims at providing an algorithm to be applied on the previous tree so that a deeply connected one is obtained. Section 5 deals with imposing a k-partite graph on the associated graph of the previously obtained clusters.

3 Imposing a Tree

In this section how using merges to impose a tree on a graph is considered. In [BM04] a merge yielding a tree is introduced. We show in this section that this merge, denoted M_1 in what follows, preserves the hierarchy introduced by the distance function from a given vertex and yields a tree with maximal number of vertices. M_1 is actually defined from an equivalence relation $\mathbf{R}$ and is then shown to merge as few vertices as possible to result in a tree (see theorem 1).

If G is an undirected simple and non-empty graph and r is a vertex of G, the relation $\mathbf{R}$ on V_G according to r is defined as the following:

$$x\mathbf{R}y \iff (d(x) = d(y)) \wedge \big(\exists z, p, p', p_{x,z} = p \tag{3}$$

$$\wedge p'_{y,z} = p' \wedge \forall x' \in p, \forall y' \in p', d(x) \leq d(x') \wedge d(y) \leq d(y')\big)$$

Basically, two vertices are related according to $\mathbf{R}$ if there exists a path from each of them leading to the same vertex by only traversing vertices that are further away from r. Considering that r would be the root of a tree, then the distance function from r would represent the depth of each vertex in this tree. Two vertices would then be related by $\mathbf{R}$ if there exists a vertex whose depth is greater and from which there is a path to both of them. The idea is then to merge iteratively two such vertices together into a single one to obtain a tree.

Note that the relation $\mathbf{R}$ is reflexive and symmetric. This implies that the transitive closure $\mathbf{R}^+$ of $\mathbf{R}$ is an equivalence relation over the set of vertices of $\mathcal{G}$. By definition, $x\mathbf{R}^+y$ if and only if there exists a sequence $(x_i)_{1\leq i\leq k}$ such that $x\mathbf{R}x_1$, $x_k\mathbf{R}y$ and for all $i \in \{1,\ldots,k-1\}$, $x_i\mathbf{R}x_{i+1}$.

It is possible to compute the classes of equivalence associated to $\mathbf{R}^+$ by searching for pairs of vertices related to each other by $\mathbf{R}$. The idea would then be to merge all the vertices of a same class to obtain a tree. Considering Figure 1 with $d_{G,r}(r) = 0$, $d_{G,r}(x) = d_{G,r}(y) = d_{G,r}(w) = 1$, $d_{G,r}(x') = d_{G,r}(y') = d_{G,r}(w') = 2$ and $d_{G,r}(z) = 3$, the corresponding classes of equivalence are $\{x,y,w\}$, $\{x',y'\}$, $\{w'\}$ and $\{z\}$. An algorithm implementing the merge M_1 can be found in [BM04].

Definition 3 Let G be a simple undirected non-empty graph and let r be a vertex of G. $M_1(G,r)$ denotes the merge defined by:

- $V_{M_1(G,r)} = \{\text{classes of equivalence of } \mathbf{R}^+\}$
- $E_{M_1(G,r)} = \{(X,Y) \in V^2_{M_1(G,r)} | \exists x \in X, y \in Y, (x,y) \in E_G\}$

As an example, the graph of Figure 3(a) represents $M_1(G,r)$ where G is the graph of Figure 1. The operator defined by Definition 3 is clearly a merge according to Definition 1. Moreover, $M_1(G,r)$ is a tree rooted in $\{r\}$ since no path from two vertices at the same level can lead to a deeper vertex. It is also worth noting that each vertex of the obtained tree $M_1(G,r)$ consists of a set of vertices of G. Moreover, given $X \in V_{M_1(G,r)}$, if $r \in R$ then

$$\forall x \in X,\ d_{M_1(G,r),R}(X) = d_{G,r}(x)$$

This equation implies in particular that M_1 preserves the hierarchy according to r.

Theorem 1 is actually the main result of this section. It shows that M_1 is a merge which yield a tree whose number of vertices is maximal, while preserving the hierarchy introduced by the distance function.

Theorem 1 *Let G be a undirected simple non-empty graph and r be a vertex of G. $M_1(G,r)$ is the maximal (in the sense of the number of vertices) tree rooted in r, preserving the hierarchy, and obtained from G by merging.*

Proof. (proof of Theorem 1)
If $M_1(G,r)$ was not an optimal tree, then there exists an optimal merge M such that $M(G)$ is a tree as well as two different vertices X and Y of $M_1(G,r)$ and two vertices x and y of G such that $x\mathbf{R}^*y$ but $x \in X$ and $y \in Y$. Since the merge M also preserves the hierarchy, $d_{M(G),R}(X) = d_{M(G),R}(Y)$ holds. Now since $x\mathbf{R}^*y$, then it exists a path between x and y such that only vertices, whose depth is greater than $d_{G,r}(x)$, are traversed. Now according to Lemma 1, there exists such a path

between X and Y in $M(G)$. In that case $M(G)$ can not be a tree. This contradicts the assumptions and so $M_1(G,r)$ is not optimal.

□

4 On the Computation of a Deeply Connected Tree

The tree obtained after applying merge M_1 to a graph does not necessarily yield a tree whose vertices abstract connected sub-graphs of G. In this section a merge denoted M_2 is then introduced to achieve this goal. This merge is performed on $M_1(G,r)$. The idea is to merge couples of vertices for which the associated sub-graphs are not connected in the initial graph.

Definition 4 Given a graph G and a tree T obtained applying a merging operator M on G, M_2 is the maximal (in number of vertices) merging operator such that

$$\forall X \in V_{M_2(T)}, \forall x,y \in V_G, x,y \in X \Rightarrow$$
$$\exists \text{ a path } p \text{ in } G \text{ between } x \text{ and } y$$

The following algorithm gives a way to merge the vertices so that for each vertex of the obtained tree, its associated sub-graph is connected.

Algorithm 1

- **Input:** a tree T obtained by applying merge M_1
- **Output:** a tree T' which the graph associated to any vertex is connected.
- Determine for each vertex of T if the associated graph is connected (denoted 1-vertex) or not (denoted 0-vertex).
- From the lower levels to the upper one,
  (1) Merge the 0-vertices with its parent.
  (2) performs action 1 till the obtained tree is deeply connected.

This principle relies on the fact that if two vertices x and x' are merged to yield a vertex whose associated graph is connected, then x is the parent of x' or vice-versa. If it was not the case then any vertex of n would not be a neighbor of a vertex of x' in the initial graph. This implies that more merges should be performed to obtain a connected graph associated to the vertex $\{x,x'\}$.

It is also worth noting that if x is the parent of x' then the graph of the vertex $\{x,x'\}$ resulting from their merging is not necessarily connected. But this becomes true if the graph associated to x is connected (because x is the parent of x' and then each vertex of x' is connected to a vertex of x).

Theorem 2 *Algorithm 1 implements Merge M_2 and then yields a deeply connected tree with maximal number of vertices, when applied to $M_1(G,r)$.*

Proof. First, note that Algorithm 1 always terminates. This holds because the merging applied in action 1 provides a tree whose number of nodes is strictly lower than the initial tree. Moreover the tree, obtained from G by merging and containing only one node, is deeply connected.

Note also that Algorithm 1 always yields a tree. This is due to the fact that the merges performed in action 1 correspond to edges-contraction and cannot then add cycles to the tree.

The optimality of the solution provided by Algorithm 1 is ensured by two features:

- First, any 0-vertex has to be merged with an other node.
- Moreover, according to the Definition 3, any vertex x of G is either a neighbor of vertices belonging to the same vertex X of $M_1(G,r)$ as x, or belonging to a neighbor of X. This means that any 0-vertex has to be merged with its parent or one of its children. If a 0-vertex is merge with one of its children, then the obtained node will be a 1-node, but the number of 1-node will remain the same as in the previous tree. If now a 0-vertex is merge with its parents:
  - if the parent is a 1-vertex, the case is similar as the first one: the obtained vertex is a 1-node and the number of 1-node remains the same.
  - If the parent is a 0-node, it might happen that the resulting node is a 1-node and then the number of 1-nodes would increase.

  We can then deduce that merging a 0-node with its parent is more beneficial to obtain an optimal solution.

□

Figure 3(d) shows the resulting tree obtained by applying successively merges M_1 and M_2 to the graph G given in Figure 1, considering Vertex r as a root for the imposed tree. According to Theorem 2, this tree is the maximal deeply connected one and obtained by merging from G, considering r as a root. Moreover, M_2 clearly preserves the hierarchy since only edge-contractions are performed.

5 Imposing a k-partite Graph on an Associated Graph

When applying the merges considered in section 3 and 4, the initial graph is clustered so that a tree is obtained. This provides greater readability which is particularly relevant if the initial graph is dense and possesses a large number of vertices. But it is also of great interest for the user to visualize what is inside a cluster, *i.e* the associated graph. Since the initial graph can be dense and large, so may be the associated graphs. It is then of interest to apply merges to clusters and then offer a more readable visualization of the associated graphs. However, imposing a tree on them which preserves the hierarchy is not possible. There are indeed at least two nodes with minimal depth, in each cluster. There is then no legitimate root to form the basis to impose a tree.

In this section k-partite graphs are considered to be imposed on the associated graphs. Indeed it can be shown that it is possible to cluster each associated graph so that a k-partite graph, with

maximal number of vertices and preserving the hierarchy, is obtained. With that aim, the merge M_3 is introduced.

Definition 5 Let G be a non empty connected undirected graph and r be a vertex of G. Let us denote G' an associated graph of $M_2(M_1(G,r))$. Merge M_3 is defined such that

- $V_{M_3(G')} = \{A \subseteq V_G | \ \forall x,y \in A,\ (x,y) \in E_G \text{ and } d_{G,r}(x) = d_{G,r}(y)\}$
- $E_{M_3(G')} = \{(X,Y) \in V^2_{M_3(G')} | \ \exists x \in X \text{ and } y \in Y \text{ with } (x,y) \in E_G\}$

As shown by Theorem 3, applying merge M_3 to each cluster of $M_2(M_1(G,r))$ yields a k-partite graph preserving the hierarchy according to r with maximal number of vertices.

Theorem 3 *Let G be a non empty connected undirected graph and r be a vertex of G. Let us denote G' an associated graph of $M_2(M_1(G,r))$. $M_3(G')$ is a k-partite preserving the hierarchy of G according to r with maximal number of vertices.*

Proof. First, let us introduce sets $(V_i)_i$ of vertices of G' such that x and y are in the same set V_i if and only if $d_{G,r}(x) = d_{G,r}(y)$. The only edges preventing G' from being a k-partite graph according to sets V_i are the ones between two edges of the same depth. But merge M_3 merges vertices so that these edges are hidden in a cluster. M_3 clearly preserves the hierarchy according to r since only vertices of the same depth are merged. Plus the sets $(V_i)_i$ are defined such that they reflect this hierarchy, partitioning the vertices according to their depth. Merge M_3 is then optimal since only vertices linked to an other vertex belonging to the same set V_i are merged. □

Theorem 3 offers a way to impose a k-partite graph structure on each associated graph. Referring to Figure 1 and only considering the levels 1, 2 and 3, the resulting graph is not a k-partite graph since x and y are neighbors and on the same level. The merge operation M_3 results in the same graph except that x and y are merged.

The hierarchy introduced in Section 2 to impose a tree is preserved. This ensures that users are not confused when entering into details and reading the contents of a cluster. Since the clustered associated graphs are k-partite, a layered layout can be applied so that two vertices of the same layer are not neighbors. But the readability of the obtained k-partite graphs is even better in this case than a typical k-partite graph. If the chosen layering consists in gathering in the same layer the vertices with same depth, then no edge crosses any layer. This property comes from the fact that the layers are defined according to the depth of the vertices.

6 Example: Exploration of Intra-Cellular Signaling Cascades

The merge operations described in the previous sections are quite general in nature and so their primary use is for the purpose of exploration with little *a priori* knowledge about the general structure of the graphical data set being investigated. A question that might be asked of such data is to what extent the graphical data set adheres to a particular structure. In the context

of the above merge operations it is useful to ask to what extent a data set adheres to a hierarchical structure. Examples of such data sets are protein-protein interaction networks that describe the observed interactions between proteins resulting from biological experimental methods (e.g [FB01, MHP01]). In this section we will describe the properties of such data sets, what sort of questions biologists may ask of such data sets and how imposing a hierarchical structure as described in the previous sections on such data can assist in understanding the underlying structure of the data.

6.1 Properties of Protein-Protein Interaction Data

Molecular interactions play an important role in determining the behavior of cells. The involved molecules can be proteins and the study of protein interactions helps to understand activities such as differentiation, development and proliferation of cells. Protein-protein interaction (PPI) networks are graphical data sets that describe the observed interactions between proteins and other molecules using particular experimental methods. Proteins and other molecules (e.g. Ca^{2+}) correspond to the nodes of the graph while observed interactions correspond to edges. Proteins interact with each other typically to alter the behaviour of another protein. Many of the interactions in these networks are well documented and have been confirmed repeatedly through various experiments. Some sequences of interactions are well known and play a direct role in particular cell behaviors. These sequences are called *pathways*. Examples of such pathways include the MAPK (involved in the cell proliferation process) and JNK-c-JUN (responding to cell stress) pathways. However, other less well understood proteins are involved in controlling these pathways. These proteins are known as regulatory or scaffolding proteins. An example of such a protein is Ste5p [GFT+01] which is found in yeast (*S.cerevisiae*). The absence of Ste5p results in the yeast cell becoming sterile. Regulatory proteins are also involved in coordinating activity between various pathways.

It is important to know, given the current data, to what extent we can understand the behavior of these networks. The development of new experimental methods to detect possible protein-protein interactions has vastly increased the biologist's understanding of how cells function. However these *high-throughput* techniques for detecting interactions can result in many false negatives and positives. While there is an abundance of well organized and easily accessible interaction data sets [ZPQ+02, MAA+, cpa] there is little additional information associated with the interactions such as level of confidence, locus or dynamics of reactions with which to judge the authenticity of such interactions. These experimental methods also rarely reproduce the same results which results in data sets from each experiment having little overlap. The ultimate goal of these experiments is to produce a mechanistic understanding of the activities in these networks. As it stands however there is little scope to achieve such a result given the current data sets and so it is best to focus on the structural properties of such networks.

6.2 Possible Queries on PPI Data

Given that we are restricted to discussing structural properties of protein interaction networks, what sort of questions can a biologist ask of such data? Often the questions are graph theoretic in nature. For example, a set of proteins interacting with each other corresponds to a clique in

the network. Existence of such structures allows the investigator to abstract and simplify the network [BZC+03].The biologist may also try to infer behaviour or roles from the structural evidence. An investigator may ask which interactions are core to a particular pathway and which are regulatory or otherwise. Another common approach is to apply understanding of the structure of one interaction network such as yeast to the structure of a less well understood network such as the human cell [GFT+01].

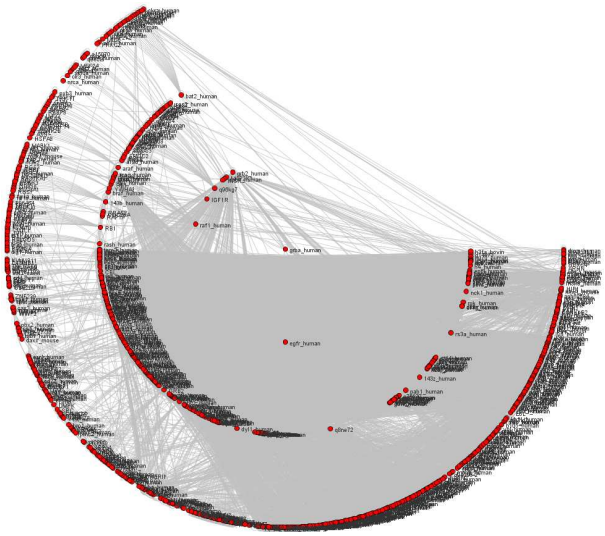


Figure 3: The initial protein interaction network before imposing a tree

6.3 Applying Hierarchy to PPI data

Our hierarchical merge operations can be applied to analyze the structure of protein interaction networks.It is natural to assume in many cases that a hierarchical structure describes the sequence of interactions in the cell. For example when the human growth hormone receptor (grba_human) is activated on the surface of the cell, many possible sequences of interactions are activated. Of interest is where the tree assumption is violated, indicating the presence of regulatory proteins. One can imagine a tree-like cascade of possible interactions with the activated human growth hormone receptor as the root. The first merge operation, M_1, can be used to induce such a structure on the data. It is a reasonable argument, given the data, that the resulting unconnected clusters are not meaningful protein interaction clusters. To produce biologically more meaningful (connected) clusters the merge M_2 is performed. Finally if the user wishes to investigate the structure inside a cluster with respect to the chosen root, merge M_3 produces a suitable abstraction.

6.4 Visualizing Protein Interaction Networks

We represent the protein-focused visualization using the standard node-link representation. Clusters are either represented as:

- An ellipse whose size is proportional to the number of underlying vertices it contains if all vertices are on the same level
- An ellipse containing a smaller radial drawing of the underlying graph whose underlying vertices are clustered using the first merge operation

We use a radial tree layout [BETT99] since it handles broad, shallow trees quite well. The user may focus on visualizing the underlying graph within each cluster by either drawing the underlying graph using a force-directed layout [BETT99] if all vertices are on the same level of the underlying hierarchy or as a layered Sugiyama style drawing [KM81] if the underlying vertices are found on multiple levels of the underlying hierarchy in which connected vertices on the same level are clustered together to create a k-partite graph.

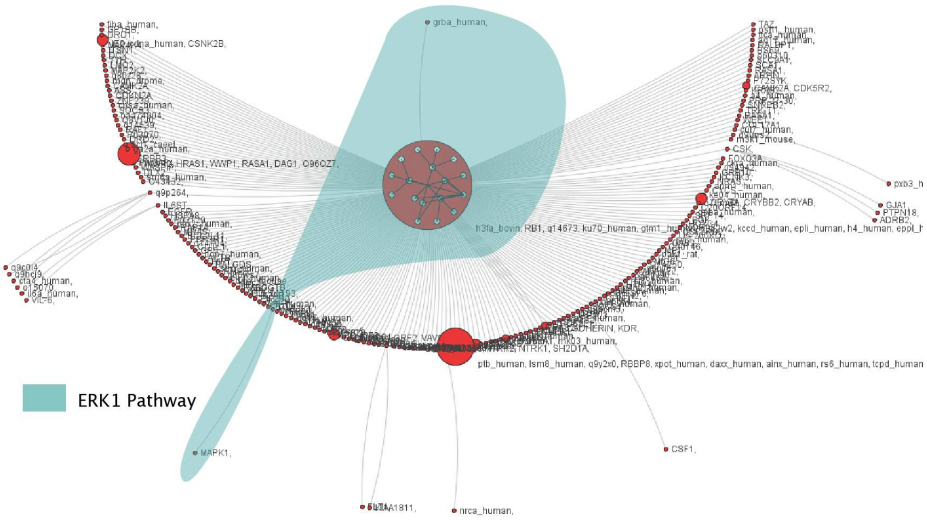


Figure 4: Imposing a tree on a protein interaction network

6.5 Data Source

We retrieved the data from the CPath protein interaction database using the protein human growth factor receptor-bound protein 10 as the focus protein. We constructed the underlying graph using vertices that were at most three interactions away from the focus protein. Our queries to the CPath web service were of the form:

http://cbio.mskcc.org/cpath/webservice.do?version=1.0&cmd=get_by_interactor_id&
q=CPATH_ID&format=psi_mi&startIndex=0&organism=9606&maxHits=50

This resulted in a graph containing 875 vertices. The imposed tree algorithm was applied to the underlying graph resulting in a clustered graph containing 210 vertices. The resulting graph is displayed in figure 4.

6.6 Discussion

The user is presented with in interface to scroll, zoom and investigate clusters in the graph. There is also a search facility to identify where a particular protein is in the visualization. Upon investigation it was found that there is no regulatory protein interacting with all proteins of the ERK1 pathway (involving the MAPK1 protein). Regulation of the ERK1 pathway, based on the evidence provided, occurs higher up the pathway. ERK1 is involved with cell proliferation. It is known from the study of the yeast cell that there exist what are known as scaffolding proteins, such as Ste5p, that interact with all parts of the ERK1 pathway in yeast and help regulate it. Such a protein does not exist for the human ERK1 pathway. So figure 4 shows that the behavior of the human cell is different from the behavior of the yeast cell, at least regarding proliferation.

In addition a significant number of proteins particularly on the leaf nodes of level 2 in figure 4 could be identified as being potentially spurious if they interact with only one other protein in the network and their function is unclear. Other spurious interactions have been absorbed into various clusters and investigation inside clusters aids in the further refinement of the tree.

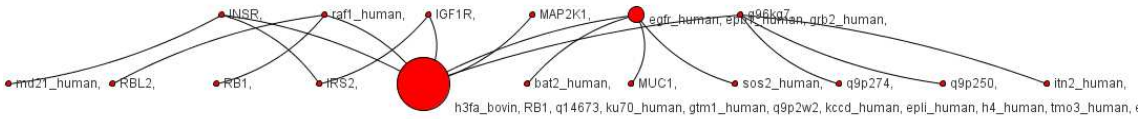


Figure 5: Inside a cluster

7 Conclusion

This study introduces a method to impose a tree on a graph by merging vertices. This method consists of two steps. The first step provides a tree with maximal number of vertices while preserving the hierarchy. The second step is meant to be applied to the previous tree. It provides a new tree whose nodes are associated to connected part of the initial graph. The composition of these two steps then yields a tree with maximal number of nodes such that the associated graphs are connected. Moreover the contents of each nodes of the tree, is itself clustered to result in a k-partite graph preserving the hierarchy introduced by the imposed tree.

We applied our approach to visualize protein interactions in human cells. It has been shown that although some proteins regulate all parts of the proliferation pathway of the yeast cell, there is no evidence of such a protein playing a similar role in the human cell. In the process of applying graph transformations to this data we discovered that dealing with biological data poses significantly different challenges when compared to transforming software structure for example. The very fact that the data is inherently uncertain requires that graph transformation techniques need to handle precision and certainty if they are to be relevant to transforming such data.

As future work, other structures to impose on the initial graph could be considered. Since a graph that does not possess a significant hierarchical structure is likely to possess a grid like structure. So imposing grids on a graph should be considered. Alternatively, we could consider extending the hierarchical notion further by investigating imposing poly-trees on the initial graph.

Acknowledgements: This work is supported by the Irish Research Council for Science, Engineering and Technology: funded by the National Development Plan in association with Microsoft Research. This work is also partly supported by the Marie Curie International Reintegration grant.

Bibliography

[BETT99] G. D. Battista, P. Eades, R. Tamassia, I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[BM04] F. Boutin, M.Hascot. Focus Dependent Multi-level Graph Clustering. *Proceedings of the Conference on Advanced Visual Interface, AVI04*, 2004.

[Bod93] H. L. Bodlaender. A Tourist Guide through Treewidth. *Acta Cybernetica* 11:1–21, 1993.

[BZC+03] D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, L. Zhang, G. Li, R. Chen. Topological structure analysis of the protein-protein interaction network in budding yeast. *Nucleic Acids Research* 31(9):2443–50, May 2003.

[cpa] Cancer Pathway Database: http://cbio.mskcc.org/cpath.

[Die05] R. Diestel. *Graph Theory*. Springer-Verlag Heidelberg, 2005.

[DK05] R. Diestel, D. Kühn. Graph minor hierarchies. *Discrete Appl. Math.* 145(2):167–182, 2005.
doi:http://dx.doi.org/10.1016/j.dam.2004.01.010

[FB01] S. Fields, P. Bartel. The two-hybrid system. A personal view. *Methods Mol. Biol.* 177:3–8, 2001.

[Fen97] Q. Feng. *Algorithms for Drawing Clustered Graphs*. PhD thesis, University of Newcastle, 1997.

[GFT+01] G.Pearson, F.Robinson, T.B.Gibson, B.-E. Xu, M.Karandikar, K.Berman, M.H.Cobb. Mitogen-activated protein (MAP) kinase pathways : Regulation and physiological functions. *Endocrine reviews (Endocr. rev.)* 22, no2:153–183, 2001.

[HHP05] W. Huang, S. Hong, P.Eades. Layout effects on sociogram perception. *Graph Drawing*, pp. 262–273, 2005.

[KM81] S. K.Sugiyama, M.toda. Methods for visual understandins of hierarchical system structures. *IEEE Transactions on Systems, Man. and Cybernetics, SMC-11(2)*, pp. 109–125, 1981.

[LKK06] K. Lehmann, S. Kottler, M. Kaufmann. Visualizing Large and Clustered Networks. *Graph Drawing 06*, Septembre 2006.

[MAA+] H. W. Mewes, C. Amid, R. Arnold, D. Frishman, U. Gldener, G. Mannhaupt, M. Mnsterktter, P. Pagel, N. Strack, V. Stmpflen, J. Warfsmann, A. Ruepp. MIPS: analysis and annotation of proteins from whole genomes. *Nucleic Acids Res*, January.

[MHP01] M. Mann, R. Hendrickson, A. Pandey. Analysis of proteins and proteomes by mass spectrometry. *Ann. rev. Biochem.* 70:437–473, 2001.

[ZPQ+02] A. Zanzoni, M. L. Palazzi, M. Quondam, G. Ausiello, H. M. Citterich, G. Cesareni. MINT: a Molecular INTeraction database. *FEBS Lett* 513(1):135–140, February 2002.